Key Action Operators in the Observable Class (Part 1)

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University

Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Recognize key operators defined in—or used with—Observables
  - Factory method operators
  - Transforming operators
  - Action operators
    - These operators don't modify an Observable, but instead use it for side effects
      - e.g., doOnNext()

Key Action Operators in the Observable Class

Key Action Operators in the Observable Class

- The doOnNext() operator
  - Add a behavior triggered when an Observable emits an item

```
Observable<T> doOnNext
  (Consumer<? super T> onNext)
```

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#doOnNext

Key Action Operators in the Observable Class

- The doOnNext() operator
  - Add a behavior triggered when an Observable emits an item
    - The behavior is passed as a Consumer param that's called on successful completion

```
Observable<T> doOnNext
  (Consumer<? super T> onNext)
```

Interface Consumer<T>

Type Parameters:

T - the type of the input to the operation

All Known Subinterfaces:

Stream.Builder<T>

Functional Interface:

This is a functional interface and can therefore be used as the assignment target for a lambda expression or method reference.

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/functions/Consumer.html

Key Action Operators in the Observable Class

- The doOnNext() operator
  - Add a behavior triggered when an Observable emits an item
    - The behavior is passed as a Consumer param that's called on successful completion
      - i.e., it is a "callback" that typically has a "side-effect"

```
Observable<T> doOnNext
  (Consumer<? super T> onNext)
```

See en.wikipedia.org/wiki/Callback_(computer_programming)

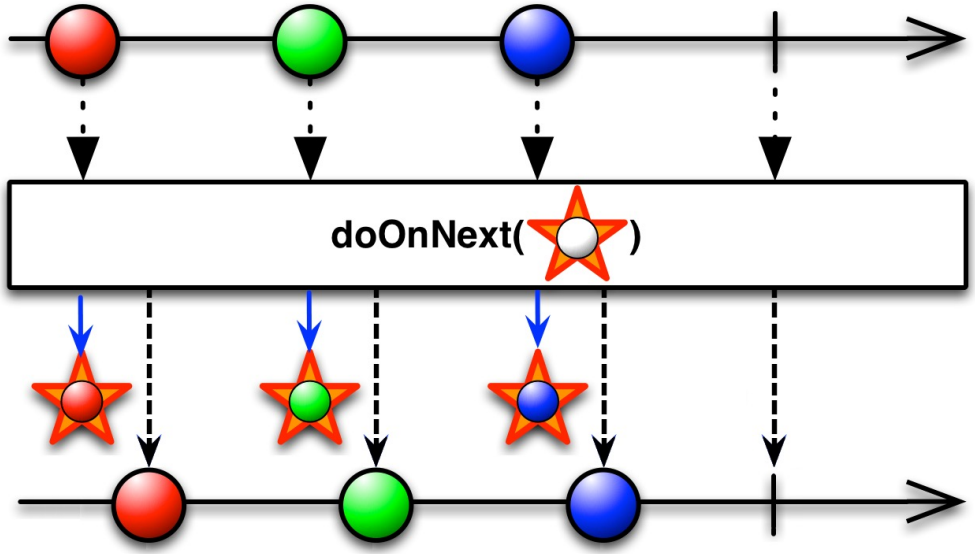
Key Action Operators in the Observable Class

- The doOnNext() operator
  - Add a behavior triggered when an Observable emits an item
    - The behavior is passed as a Consumer param that's called on successful completion
  - Returns an Observable that is not modified at all
    - i.e., the type and/or value of its elements are not changed

```
Observable<T> doOnNext
  (Consumer<? super T> onNext)
```

Key Action Operators in the Observable Class

- The doOnNext() operator
  - Add a behavior triggered when an Observable emits an item
  - Used primary for debugging, logging, and/or getting visibility into an Observable chain



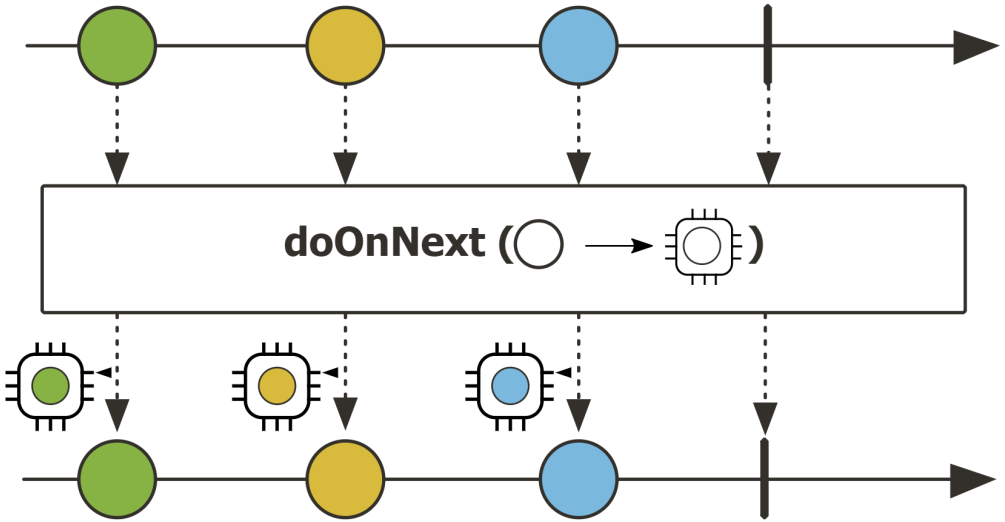
Log each BigFraction value on success (otherwise skip)

```
Observable
   .fromIterable(bigFractionList)
   .doOnNext(bf ->
                logBigFraction(sUnreducedFraction, bf, sb))
   ...
```

See Reactive/Observable/ex1/src/main/java/ObservableEx.java

Key Action Operators in the Observable Class

- The doOnNext() operator
  - Add a behavior triggered when an Observable emits an item
  - Used primary for debugging, logging, and/or getting visibility into an Observable chain
- Project Reactor's operator Flux .doOnNext() works the same




```
Flux
   .fromIterable(bigFractionList)
   .doOnNext(bf ->
             logBigFraction(sUnreducedFraction, bf, sb))
   ...
```

Log each BigFraction value on success (otherwise skip)

See projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#doOnNext

Key Action Operators in the Observable Class

- The doOnNext() operator
  - Add a behavior triggered when an Observable emits an item
  - Used primary for debugging, logging, and/or getting visibility into an Observable chain
  - Project Reactor's operator Flux .doOnNext() works the same
  - Similar to Stream.peek() method in Java Streams

peek

`Stream<T> peek(Consumer<? super T> action)`

Returns a stream consisting of the elements of this stream, additionally performing the provided action on each element as elements are consumed from the resulting stream.

This is an intermediate operation.

For parallel stream pipelines, the action may be called at whatever time and in whatever thread the element is made available by the upstream operation. If the action modifies shared state, it is responsible for providing the required synchronization.

```
List<String> collect = List
   .of("a", "b", "c").stream().peek(System.out::println)
   .map(String::toUpperCase).toList();
```

See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#peek

End of Key Action Operators in the Observable Class (Part 1)